

**REMARKS**

Claims 1 to 39 are pending in the present application. Claims 1, 5 and 8 are the independent claims. Claims 1, 5 and 8 were amended herein to emphasize that the solutions enabled by the present invention were developed for a distributed computing environment. Support for the amendments can be found at least from page 11, line 11 to page 20, line 19. No new matter was added.

In the Office Action, dated June 16, 2003, claims 1-2, 4-5, 7-13, 20-23 and 37-39 were rejected under 35 U.S.C. § 103(a) over U.S. Patent No. 6,438,618 (Lortz et al.). Claims 3, 6, 27 and 35 were rejected under 35 U.S.C. § 103(a) over Lortz et al. in view of U.S. Patent No. 5,857,190 (Brown). Claims 14-15 and 19 were rejected under 35 U.S.C. § 103(a) over Lortz et al. in view of U.S. Patent No. 6,363,435 (Fernando et al.). Claims 16-18, 24-26 and 36 were rejected under 35 U.S.C. § 103(a) over Lortz et al. in view of U.S. Patent No. 6,446,136 (Pohlmann et al.). Claims 28-34 were rejected under 35 U.S.C. § 103(a) over Lortz et al. in view of Brown and further in view of Pohlmann et al. The outstanding rejections to the claims are respectfully traversed.

**Formal Matter – Priority Claim**

In the Office Action, dated June 16, 2003, Applicants note that the Office has not yet acknowledged Applicants' priority claim to U.S. Provisional Application No. 60/118,668, filed February 3, 1999. Acknowledgement of Applicants' priority claim is respectfully requested.

**Summary of the Invention**

The invention provides a generalized tracking system for a distributed computing environment, such as a peer to peer computing environment, which provides for tracking when a software component changes state and provides corresponding state change notification when a tracked software component changes state. Since the architecture of the present invention was designed for a distributed computing environment, as will be illustrated below, there are many features of the invention that distinguish over the art of record.

**Distributed Tracking System**

In one embodiment, the object tracking system provides an asynchronous notification mechanism for notifying a client, when an object that is referenced by the client changes state in a distributed object environment. An object can be either in an up state (*e.g.*, instantiated) or a down state (*e.g.*, destructed). The object tracking system also provides a mechanism through which a client can register its interest in accessing an object, can retrieve a pointer to the object when the object is in the up state, and can unregister its interest in accessing the object. The object tracking system interacts with an object manager that provides the facility to locate objects, to monitor the state of objects, and to retrieve pointers to the objects. The object manager notifies the object tracking system when objects change state.

In one embodiment, the watching of a resource is coordinated by the bus manager, but the monitoring of a resource is performed on a client-to-server node basis without interaction from the bus manager. When a client wants to watch a resource so that it knows when the resource is in the up state, the client node notifies the bus manager. The resource is identified using a tracking reference. If the resource is already up, then the bus manager then notifies the client node that the resource is up. Otherwise, the bus manager notifies the client node when the resource comes up. When the client node is notified that the resource is up, it may notify the bus manager to stop watching for the resource.

The monitoring of a resource is performed on a peer-to-peer basis. That is, once a client node is informed that a resource has entered the up state, it establishes a connection directly with the server node that contains the resource. Once the connection is established, the client node notifies the server node periodically that it is still up and running. If the server node does not receive this notification, it assumes that the client node is no longer up and running and resets its internal state accordingly. Similarly, if the client node receives an error when sending its periodic notification to the server node, it assumes the server node is down and resets its internal state accordingly. When the resource goes down, the server node notifies the client node that the resource is now down.

Each node includes clients 205, a resource tracking system 206, and a resource manager 207. A client requests the resource tracking system to provide pointers to resources and to notify the client when resources come up or go down. The resource tracking system interacts with the resource manager to watch, monitor, and retrieve pointers to the resource.

The resource manager also detects when resources on its node come up and go down and notifies the bus manager or other nodes as appropriate. See, e.g., page 15, line 13 to page 16, line 11.

Property Notification System

Independently for use in a distributed computing system, the invention provides a property notification system.

In one aspect, the invention provides components on the client side to support the watching of properties of a server resource. When a client resource wants to watch a property of a server resource, the client resource registers to track the server resource. When a property of a server resource is being watched, a special type of client object is added to the list of client objects of the resource reference object for that server resource. The special type of client object indicates that it represents the watching of a certain property and includes a property reference object 5404 for each time that the client resource has registered to watch that property. A client resource also includes a synchronize property function 5406 and a property set function 5407. The synchronize property function is invoked by the server resource when a client resource first registers to watch a certain property of that server resource to provide the current value of the property to the client resource. The property set function is invoked whenever a watched property is set. See, e.g., Fig. 54 and accompanying description.

In another aspect, the invention provides components on the server side to support the watching of properties of a server resource. A server resource 5501 includes a property/client table 5502. The property/client table contains an entry for each property of the server resource that is being watched. Each entry contains the name of the property, the value for that property, and a client watching object 5503 for each client resource that has registered to watch that property. Each entry may also include a queue for storing property values in the order in which they are set pending notification of each of the client resources. A server resource also includes a watch property function 5504 and a stop watching property function 5505. The watch property function is passed an indication of a property of the server resource, the identification of the client resource, and a context. The watch property function adds a client watching property object in the property/client table to indicate that the client

resource is now watching the property. The watch property function also requests that the client resource to be monitored using a monitoring component 5506. See, e.g., Fig. 55 and accompanying description.

*Event Notification System*

Independently for use in a distributed computing system, the invention provides an event notification system.

The event system provides a mechanism for providing event notifications when events are generated by resources. An event is an asynchronous signal that is distributed to all client resources, also referred to as listeners, who have registered to listen for an event signal. In one embodiment, the event system neither guarantees that a listener will receive the events in the order they are generated nor guarantees that each listener will receive every event for which it is listening. Each event has an associated event type. A listener registers to listen for events of a certain event type. In one embodiment, the event types may be hierarchically organized. For example, one event type may be a timer event. The timer events may be further classified into catastrophic timer events, warning timer events, and informational timer events, which are sub-events. An informational timer event may further be classified into start-up timer events and shut-down timer events. A listener may register to listen for events at any level in the event hierarchy. For example, a listener may register to listen for informational timer events. That listener would receive an event notification as for start-up timer events and a shut-down timer events. A listener will receive event notifications for leaf events of the sub-tree correspond to the event type registered. A leaf event is an event that is not further classified into sub-events. An event type may have its hierarchy embedded in its name. For example, the name of start-up timer event may be “/timer event/informational time event/start-up timer event.”

A client 6301 registers to listen for events by sending a listen message along with an event type to the listener component 6303. The client receives from the listener component an event notify message along with event information when an event of that event type is generated. The client un-registers its interest in listening for events of a certain event type by sending a stop listening message along with the event type to the listener component. In one

embodiment, each node as a listener component through which is routed all event related messages for all listeners on that node. The listener component may in turn route event-related messages to a listener bus manager 6305. The listener component notifies the listener bus manager to listen for all event types for which listeners on that node have registered. The listener component may send only the listener bus manager. One listen message for each event type regardless of how many listeners at that node have registered for that event type. For example, if a listener component receives requests from six clients, the listener component sends only one listen message to the listener bus manager. The listener component maintains a listener table cache 6306 that contains a mapping from each event type for which a listen request has been registered and each client that has registered for that event type. When the listener component receives an event notification, it uses the listener table cache to notify each listener that has registered for that event type. In this way, the listener component reduces the event messages that are sent between that node and the node of the listener bus manager. When the listener component receives event notifications, it queues an event notifications for each listeners. The listener component uses a separate thread for providing the event notification to each listener. If a single thread were used to notify each listener, the event notifications could be delayed to some listeners as a result of a delay or problem in notifying another listener. The use of a separate thread for each listener ensures that the notification to one listener will not be delayed as a result of an event notification to another listener. The listener component may receive a bus state change message. If the bus goes down and then comes back up, the listener component can re-register with the listener bus manager to receive the events of the event types in its listener table cache. The listener component may also optimize its sending of listen requests based on the event hierarchy. For example, if a listener registers to listen for a informational timer, the listener component will register that request with the listener bus manager. If another listener registers to listen for a start-up timer, then the listener component will not need to register that request with the listener bus manager. Since the listener component has already registered to receive a higher-level event type, it is already registered to receive all lower level event types. See, e.g., Fig. 63 and accompanying description.

The listener bus manager maintains a listener table 6307. The listener table contains a mapping from each event type to the registering nodes. When the listener bus manager

receives an event posting, it notifies each node who has registered to listen for events of that event type and any event type that is a parent event type. The listener bus manager queues event notifications in a manner that is similar to the queuing performed by the listener component. In particular, the listener bus manager allocates a different thread for each node to which an event notification is sent so that the event notifications to other nodes are not delayed because of problems in notifying one node. The listener bus manager may receive a node is down message and remove the entries from the listener table for that node so that no more event notifications will be sent to that node. A server 6302 may generate and post events by sending a post event message to the listener bus manager. The post event message includes event information that describes the event. The event information may include the event type, a time associated with the event, an indication of who generated the event, and the reason the event was generated.

Accordingly, the present invention provides a distributed tracking system having distributed tracking, property notification and event notification for distributed server and client nodes of a distributed computing environment.

Lortz et al.

Lortz et al., on the other hand, relates only to an event notification system and discloses an event notification system that has a centralized architecture, providing advantages over the disclosed prior art Fig. 2 of Lortz et al., but Lortz et al. nowhere teaches or suggests a distributed tracking system. To the contrary, the touted advantage of Lortz et al. is its central location layered between clients and device control objects.

In the prior art system of Fig. 2, a device 24 is connected through a network 10 to a control object 25 that controls the device and communicates with a client 20. In this regard, no filtering system is present and no separate event provider service is operating. See, e.g., Col. 3, lines 17-19. Another limitation of the example shown in Fig. 2 is the direct connection between control objects 25 and clients 20. See, e.g., Col. 5, lines 53-54. In addition, the client 20 must know about and subscribe to each control object 25 for each device 24 from which relevant events 22 may be signaled. There is no mechanism for the client 20 to be notified of events 22 that may be relevant, but are generated by a control object of which the client 20 is ignorant. See, e.g., Col. 6, lines 5-9. Lortz et al. continues to

disclose that it is therefore desirable to add an event provider service as a layer between the control objects 25 and the clients 20 that may act as a server with respect to the client 20.

In consideration of that desire, Lortz et al. discloses to employ an event provider server 30 and event filtering via event filters 31, wherein the event provider server 30 is in communication with the control objects 25. The server 30, by operating as an independent process between the control objects 25 and the clients 20, allows connections by the control objects 25 and the clients 20 to be made independent of each other. This means that, for example, a client 20 can subscribe to events 22 from a control object 25, even if the control object 25 is not executing at that time (i.e., the problem of initialization order dependency is relieved). Accordingly, Lortz et al. cannot be said to teach or suggest a distributed tracking system. See, e.g., Col. 6, lines 37-47.

*Outstanding Rejection under 35 U.S.C. § 103(a)*

As discussed above, Applicants respectfully submit that Lortz et al. nowhere teaches or suggests a distributed tracking system because Applicants merely understand Lortz et al. to disclose a centralized event server which performs event filtering positioned between clients and control objects.

Accordingly, Lortz et al. cannot be said to disclose or suggest at least a distributed tracking system for tracking when a software component changes state and for providing a state change notification of a change in state of the tracked software component (claims 1 and 5) or via a distributed tracking system, tracking when a software component changes state and providing a state change notification of a change in state of the tracked software component (claim 8).

Brown was cited for reasons related to a logging system, Fernando et al. was cited for reasons related to hierarchically organized event types and Pohlmann et al. was cited for reasons related to discrete or non-discrete event types, but none of Brown, Fernando et al. or Pohlmann et al. cure the above-identified deficiency of Lortz et al. with respect to Applicants' invention.

Accordingly, Applicants respectfully submit no prior art system of record, taken alone or in combination, teaches or suggests at least the above-discussed features of the present invention. Claims 2-4, 6-7 and 9-39 depend from claims 1, 5 and 8, respectively, and are

**DOCKET NO.: MSFT-0677 (183204.1)**  
**Application No.: 09/322,852**  
**Office Action Dated: June 19, 2003**

**PATENT**

believed allowable for the same reasons. Withdrawal of the rejections to claims 1-39 under 35 U.S.C. § 103(a) is respectfully requested.

**CONCLUSION**

Applicants believe that the present Amendment is responsive to each of the points raised by the Examiner in the Office Action, and submit that Claims 1-39 of the application are in condition for allowance. Favorable consideration and passage to issue of the application at the Examiner's earliest convenience is earnestly solicited.

Date: September 10, 2003



Thomas E. Watson  
Registration No. 43,243

Woodcock Washburn LLP  
One Liberty Place - 46th Floor  
Philadelphia PA 19103  
Telephone: (215) 568-3100  
Facsimile: (215) 568-3439